



White Paper

Automating the Advanced GNSS Spoofing Simulation Tutorial

Jaemin Powell

Applications Engineer, Orolia Defense & Security

Introduction

Utilizing the Skydel's Python API can provide major advantages in creating a test script. It will allow the user to quickly automate the setup of the BroadSim environment, while also allowing simpler environment management during a simulation. This paper will provide a step-by-step walkthrough on how to start the intuitive automation process and provide an example script from the Advanced GNSS Spoofing Simulation Tutorial. The script will include a summary of scripting steps and an in-depth description of the classes and functions used to communicate with Skydel.

Scenario Description

The intent of this scenario is to provide a simulated path across a bridge with two different types of threats working together with a common goal of capturing the receiver. The first threat is an AWGN (Additive White Gaussian Noise) knockout jammer. This jammer will reduce the GNSS signal strength as the receiver gets closer to it until the receiver loses lock and attempts to reacquire the GNSS signals. The second threat is a spoofer transmitter. The spoofer will be configured for stronger signals so that when the receiver is in a vulnerable state due to the knockout jammer, the receiver will be more likely to re-acquire on the spoofing signals. If the threats are successful, the receiver will follow the spoofer path and diverge from the truth simulated path.

Automation Setup

Follow the steps below to get an export of the python scripts that are needed to fully automate the script:

1. Set up the Skydel instances by following along with the [Advanced GNSS Spoofing Simulation Tutorial video](#). This provides visual step-by-step instructions to initialize the Skydel instance (BroadSim → Skydel) and the Skydel Spoofer (1) instance (BroadSim → Skydel Instances → Skydel Spoofer Instance 1).
2. In the “Automate” tab, select “Export to Python” on the Skydel instance, then on the Skydel Spoofer instance to export a fully functional python script for each instance. The standard folder location of the exported scripts is “/opt/broadSim/SDX/API/Python”. *Figure 1* and *Figure 2* show how the exported python files will look.

Script Description

When opening the exported python files, it is important to notice that these scripts replicate the content of the commands in the Automation tab of Skydel. These scripts give the user a great jump start in automating the BroadSim test.

Starting at line 5 of *Figure 1* and *Figure 2*, are the imports of files needed to execute the functions throughout the rest of the script. The * in some of the import lines command the script to import all variables, classes, methods, etc. in the file without having to prefix into them when they are trying to be referenced in the script.

Line 10 of *Figure 1* initializes an instance, named “sim”, of the `RemoteSimulator()` class (or the `RemoteSpooferSimulator()` class in *Figure 2*). “sim” can communicate with the Skydel instance or the Skydel Spoofer instance by first initializing the connection using the IP and instance ID parameters in the `connect()` command (e.g. `sim.connect(ip="localhost",id=0)`). `connect()` defaults to an IP of “localhost” and either an ID of 0 for the `RemoteSimulator()` instance or 1 for the `RemoteSpooferSimulator()` instance. ID is then incremented by 1 for each new instance needed in Skydel.

connect() Example – If a scenario is needed with two `RemoteSimulator()` instances (aka Skydel instances) and two `RemoteSpooferSimulator()` instances (aka Skydel Spoofer instances), then the `RemoteSimulator()` connection would be with an ID of 0 and 1 and the `RemoteSpooferSimulator()` connection with an ID of 1 and 2.

```

1 #!/usr/bin/python
2
3 # This Python script has been generated by the SKYDEL GNSS simulator
4
5 from datetime import datetime
6 from datetime import date
7 from skydel.sdx import *
8 from skydel.sdx.commands import *
9
10 sim = RemoteSimulator(True)
11 sim.connect()
12
13 sim.call(New(True, True))
14 sim.call(SetModulationTarget("DTA-21158", "", "", True, "{eef9883-28d8-4919-89da-b3625268eff6}"))
15 sim.call(SetModulationTarget("DTA-21158", "", "", True, "{e1e912bb-7bc9-4641-9dd3-8e48c46f5800}"))
16 sim.call(SetModulationTarget("DTA-21158", "", "", True, "{0729ef18-e4f6-4a97-931b-b8864349eb85}"))
17 sim.call(SetModulationTarget("DTA-21158", "", "", True, "{5dc21ee2-2e1b-4448-ac8e-f806e41a645e}"))
18 sim.call(ChangeModulationTargetSignals(0, 12500000, 85000000, "UpperL", "L1CA,E1", 50, True, "{eef9883-28d8-4919-89da-b3625268eff6}", None))
19 sim.call(ChangeModulationTargetInterference(0, 12500000, 85000000, 1, 1.57542e+09, 30, "{0729ef18-e4f6-4a97-931b-b8864349eb85}", "L1CA,E1"))
20 sim.call(AddSpooferTx("Spoofer 1", True, "127.0.0.1", 1, "{4bacac13-14ed-4ba4-ae8b-b5409de86299}"))
21 sim.call(EnableSpooferTx(False, "{4bacac13-14ed-4ba4-ae8b-b5409de86299}"))
22 sim.call(SetSpooferTxRefPower(2, "{4bacac13-14ed-4ba4-ae8b-b5409de86299}"))
23 sim.call(SetSpooferTxFixEcef(-2.71226e+06, -4.25582e+06, 3.88889e+06, 0, 0, 0, "{4bacac13-14ed-4ba4-ae8b-b5409de86299}"))
24 sim.call(AddIntTx("Transmitter 1", True, 1, True, 5, "{4f0eb2f7-5c82-44d1-8c16-dbd629fb4c2}"))
25 sim.call(SetIntTxAWGN(True, 1.57542e+09, 0, 3e+06, "{4f0eb2f7-5c82-44d1-8c16-dbd629fb4c2}", "{72c11a43-1ad7-428b-893c-635c4e3bd77c}", 458914186, None))
26 sim.call(SetIntTxFixEcef(-2.78871e+06, -4.25537e+06, 3.89017e+06, 0, 0, 0, "{4f0eb2f7-5c82-44d1-8c16-dbd629fb4c2}"))
27 sim.call(SetGpsStartTime(datetime(2020, 6, 21, 7, 0, 0)))
28 sim.call(SetVehicleTrajectory("Route"))
29 sim.start()
30 sim.post(EnableSpooferTx(True, "{4bacac13-14ed-4ba4-ae8b-b5409de86299}"), 174.842)
31 sim.stop(478.0)
32 sim.disconnect()
33

```

Figure 1: Export to Python of the Skydel instance.

```

1 #!/usr/bin/python
2
3 # This Python script has been generated by the SKYDEL GNSS simulator
4
5 from datetime import datetime
6 from datetime import date
7 from skydel.sdx import *
8 from skydel.sdx.commands import *
9
10 sim = RemoteSpooferSimulator(True)
11 sim.connect()
12
13 sim.call(New(True, True))
14 sim.call(SetModulationTarget("Spoofer", "", "", True, "{940f653d-8643-4ff5-9948-3b7bc8bd2c4b}"))
15 sim.call(ChangeModulationTargetSignals(0, 12500000, 100000000, "UpperL", "L1CA,E1", 0, False, "{940f653d-8643-4ff5-9948-3b7bc8bd2c4b}", None))
16 sim.call(SetGpsStartTime(datetime(2020, 6, 21, 7, 0, 0)))
17 sim.call(SetVehicleTrajectory("Route"))
18 sim.disconnect()
19

```

Figure 2: Export to Python of the Skydel Spoofer instance.

After “sim” has been connected, it can send commands to the Skydel instance, by using the `call()` method. The `call()` method will be used to send commands to the Skydel instance at a specified time until “sim” is disconnected (i.e. `sim.disconnect()`). Below is a list of the commands (with their description and the associated Skydel command) that are used by the `call()` method in *Figure 1* and *Figure 2* above:

New() – Open a new configuration based on the command inputs of discarding the current configuration and loading the default configuration.

SKYDEL COMMANDFile → **New Configuration**

SetModulationTarget() – Adds radio outputs by setting the hardware type, file path, IP address, clock and unique identifier of the radio.

SKYDEL COMMANDSettings → Output → **Add DTA-2115B**

ChangeModulationTargetSignals() – Initializes the radio with a GNSS signal selection output type, minimum and maximum sampling rate, frequency band, signal types, output gain, Gaussian Noise, and a unique identifier.

SKYDEL COMMANDSettings → Output → Radio # → Signal Selection → **Edit** → **Select GNSS, Upper L-Band or GNSS, Lower L-Band under Output Type**

To get the full script command, edit Radio 1 and under Signal select GPS L1 C/A and Galileo E1.

ChangeModulationTargetInterference() – Initializes the radio with a GNSS signal selection output type, minimum and maximum sampling rate, frequency band, signal types, output gain, Gaussian Noise, and a unique identifier.

SKYDEL COMMANDSettings → Output → Radio # → Signal Selection → **Edit** → **Select Interference under Output Type**

To get the full script command, edit Radio 3 to the following:

- Under Output Type, select Interference
- Under Central Frequency, check Choose with signal selection and select GPS L1 C/A and Galileo E1.

AddSpoofTx() – Adds radio outputs by setting the hardware type, file path, IP address, clock and unique identifier of the radio.

SKYDEL COMMANDSettings → Spoofers → **Add Spoofer**

EnableSpoofTx() – Enables or disables the spoofer transmitter using an enable status and the spoofer transmitters unique identifier.

SKYDEL COMMANDSettings → Spoofers → [Spoofer Name] → General → **Enabled**

To get the full script command, uncheck Enabled.

SetSpoofTxRefPower() – Sets the reference power (dBm) of the spoofer transmitter using the reference power and the spoofer transmitter's unique identifier.

SKYDEL COMMANDSettings → Spoofers → [Spoofer Name] → General → **Reference Power**

To get the full script command, set the Reference Power to 2.00 dBm.

SetSpoofTxFixEcef() – Sets the fixed ECEF position (m) and orientation (rad) of the spoofer transmitter using the ECEF x, y and z position, the yaw, pitch and roll orientation and the spoofer transmitter's unique identifier.

SKYDEL COMMANDSettings → Spoofers → [Spoofer Name] → General → **Trajectory → Fixed Edit**

Skydel uses the LLA position and converts the position to ECEF coordinates. To get the full scrip command, set the LLA position to:

- Latitude = 37.80188803°
- Longitude = -122.51448154°
- Altitude = 2 m

AddIntTx() – Adds the interference transmitter and initializes its motion type, and reference power using the name of the interference transmitter, enable status, interference group number, motion type, transmitter reference power and the interference transmitter's unique identifier.

SKYDEL COMMANDSettings → Interference → **Add Dynamic**

To get the full script command, set the Reference Power to 5.00 dBm.

SetIntTxAWGN() – Sets an AWGN interference signal to the interference transmitter by using the enable status, central frequency, power relative to the transmitter reference power, bandwidth, interference transmitter's unique identifier, AWGN signal's unique identifier, seed, and interference group number.

SKYDEL COMMANDSettings → Interference → [Interference Name] → Signal → **Add AWGN**

To get the full script command, edit AWGN jammer's bandwidth to 3 MHz.

SetIntTxFixEcef() – Sets the fixed ECEF position (m) and orientation (rad) of the interference transmitter using the ECEF x, y and z position, the yaw, pitch and roll orientation and the interference transmitter's unique identifier.

SKYDEL COMMANDSettings → Interference → [Interference Name] → **Trajectory → Fixed Edit**

Skydel uses the LLA position and converts the position to ECEF coordinates. To get the full script command, set the LLA position to:

- Latitude = 37.82553065°
- Longitude = -122.47836837°
- Altitude = 2 m

SetGpsStartTime() – Sets the simulation start date and time using the datetime python library.

SKYDEL COMMANDSettings → Start Time → **Custom Time**

To get the full script command, edit the Date to 21 Jun 2020 and the Time to 07:00:00.

SetVehicleTrajectory() – Sets ONLY the vehicle trajectory type, it does not import the route of the vehicle.

SKYDEL COMMANDSettings → Vehicle → Body → Trajectory → Select **Vehicle Simulation**

Once the Skydel instance and the Skydel Spoofer instance are completely setup, the scenario is ready to start using the `sim.start()` command. Notice that only the Skydel instance needs the `start()` and `stop()` because the `AddSpoofTx()` uses the address and instance ID of the Skydel Spoofer instance. Synchronizing these instances makes the Skydel instance the master and the Skydel Spoofer instance the slave. So, starting or stopping the simulation in the Skydel instance, starts or stops the simulation in the Skydel Spoofer instance. The `sim.stop()` command will stop the simulation at a specified amount of time. However, before stopping the simulation, other commands need to be sent in.

In this example, the spoofer transmitters needs to be enabled to capture the receiver. To send in commands during the simulation, use either the `post()` method or the `call()` method that is used for most of the script. Both commands take in a command parameter and a timestamp to send in the command. Line 30 of the Figure 1 shows an example of how the `post()` method is used by sending in the `EnableSpoofTx()` command 174.842 seconds into the simulation time. Replacing `post()` with `call()` will execute the same command. The only difference between the methods is that `call()` will output a success command that can be seen in the command prompt window when the script is executed.

Next Steps

To fully automate the spoofing test there are a couple areas that need to be updated. The first update is to reference the scripts to each other or merge them. The simplest way is to copy the Skydel Spoofer instance export and paste it to the other exported script. The `RemoteSpooferSimulator()` instance will need to be renamed from `sim` in order for the script to talk with both instances in the same script.

The second update is to define the trajectory routes of the truth and spoof signals as `.csv` files instead of the `.kml` files that were used in the tutorial. This also includes writing a function that parses through the `.csv` file and sends the information to Skydel. The section below outlines one way to create a `.csv` file using a `.kml` file in the BroadSim. In the Python API (`/opt/broadsim/SDX/API/Python`) folder there is a script, `example_create_route_csv.py` (Figure 3), that provides an example on how to parse through a `.csv` file and send the information to Skydel through an instance command. Below is a list of the commands, not described in the previous section, and the functions that are used by the `sim` instance and provided by the Python API in Figure 3:

Lla() – Instance of the `Lla()` class that contains the latitude (rad), longitude (rad) and altitude (m) position.

toRadian() – Converts value from degrees to radians.

setVerbose() – Sets the status of the verbose parameter throughout all instance commands. If true, every time a command is executed the command prompt will output a brief description of the `sim` command.

beginRouteDefinition() – When the vehicle trajectory is set to “Route”, this command initializes the `sim` instance to accept route speed and LLA or ECEF positions.

pushRouteLla() – After the `beginRouteDefinition()` command has been executed, this command can be used to send one instance of speed (m/s) and LLA position (using the `Lla()` class).

endRouteDefinition() – When the route has been defined for `sim`, the route definition can be closed and this command will output the count of nodes that were sent in since the beginning of the route definition.

NOTE: The equivalent Skydel command of the commands and functions above is in Settings → Vehicle → Body → Trajectory → Vehicle Simulation Edit → Import CSV with speed limits, Next → Select `.csv` file, Open.

```

20 FILE_PATH = 'example_route_boat.csv'
21
22 def readRow(row):
23     if len(row) < 4: raise StopIteration('Found invalid csv row at line %d' % (reader.line_num))
24     # Each row has [Speed (m/s), Lat (deg), Lon (deg), Alt (m)] as string
25     speed = float(row[0])
26     lat = float(row[1])
27     lon = float(row[2])
28     alt = float(row[3])
29     lla = Lla(toRadian(lat), toRadian(lon), alt)
30     return [speed, lla]
31
32 if not os.path.isfile(FILE_PATH):
33     print('File not found', FILE_PATH)
34     sys.exit(0)
35
36 positions = []
37
38 print("Reading " + FILE_PATH)
39
40 with codecs.open(FILE_PATH, 'rb', encoding="utf-8") as f:
41     reader = csv.reader(f)
42     field_names = next(reader) #Read CSV Header
43
44     for row in reader:
45         position = readRow(row)
46         positions.append(position)
47
48 sim = skydelsdx.RemoteSimulator()
49 sim.setVerbose(True)
50 sim.connect()
51
52 sim.call(SetVehicleTrajectory("Route"))
53
54 sim.beginRouteDefinition()
55
56 for i in range(len(positions)):
57     speed = positions[i][0]
58     lla = positions[i][1]
59     print("{0}: {1}, {2}m/s".format(i, lla, speed))
60     sim.pushRouteLla(speed, lla)
61
62 count = sim.endRouteDefinition()
63
64 sim.disconnect()

```

Figure 3: Parsing .csv files example script, example_create_route_csv.py

Along with the proper imports, the *Figure 3* example script can be used to define the route definitions for the Skydel instance and the Skydel Spoofer instance. With some adjustments, the python exports can quickly be made into a fully automated script. All that remains is to execute the script by following the steps outlined in the “**Executing a Python Script**” section. The script will then initialize both instances with the set-up steps (including the definition of the routes for the simulation and the spoofer), start the simulation, and turn on the spoofer transmitter at the specified time. This will reduce the risk of user error and provide a repeatable simulation for data analysis.

Creating a Trajectory File Using BroadSim

For this example, a trajectory **.csv** file was created from the original **.kml** file using Skydel by following the steps below:

1. Open a new configuration in the main Skydel instance.
2. Go to Settings → Output and Add one DTA-2115B Radio.
 - a. In Signal Selection, click Edit and under Signal select GPS L1 C/A and Galileo E1.
3. Go to Settings → Global → Logging and select Raw Logging (csv) at 10 Hz or 100 Hz.
4. Go to Settings → Vehicle → Body and select Vehicle Simulation.
 - a. Click Edit, select Import KML then click Next.
 - b. Click Select File and find the **.kml** file that needs to be in **.csv** format (**Spoof_Tutorial_Video_Trajectory_2.kml** or **Truth_Tutorial_Video_Trajectory_2.kml**) then select Open.
 - c. Click Next and select a speed of 80.00 km/h then click Finish.
5. Click Start and let the simulation run to completion. The **.csv** trajectory file will be in the Logging Folder. The location can be found in Settings → Global → Logging.

Example Script

The **advanced_gnss_spoofing_simulation_tutorial.txt** is an example of a python script that shows the transition that can be made from the exported files. The script has a structured arrangement to increase readability. It summarizes steps that are executed and provides a description of the classes and functions that are used to communicate with Skydel. The `Vars()` class includes the names of variables that are used throughout the script. It allows the user to easily make updates to the scenario. Each class and function in this script has a brief description of the functionality and its parameters in the docstring. After the imports, the `main()` function is the starting point for the script execution. Please begin reading through the `main()` function and the rest of the example script for more of a description of the Skydel API integration into the Python environment.

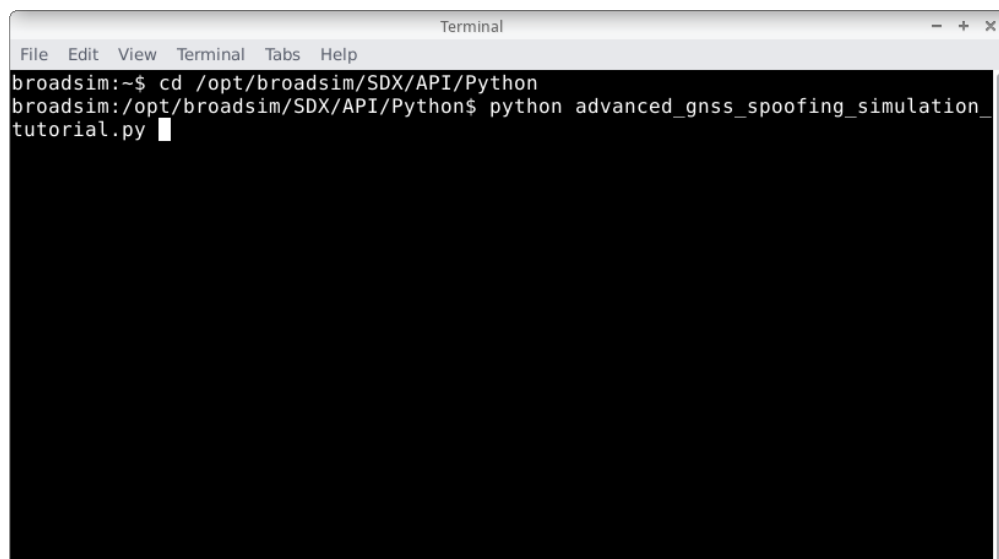
Executing a Python Script

After the python script has been completed the next step is to properly set up Skydel so the script can communicate with Skydel. All the Skydel instances that the script is going to communicate with needs to be open. For this script, the Skydel instance (BroadSim → Skydel) and the Skydel Spoofer (1) instance (BroadSim → Skydel Instances → Skydel Spoofer Instance 1) needs to be opened.

When both instances have been opened the script is ready to be executed. There are two different ways to execute the python script. The first and simplest way is to go to the python script location, right click on the **.py** file and select “Python (Execute)”. This will briefly open a command prompt window where it will print out parts of the script that have been commanded to print out as well as traceback information of scripting errors, if any, that have been caught. Unfortunately, when the script has been completed the command prompt window will close without saving any of the information that was printed out. This execution process is good to do when the script runs through without any errors.

The second way is to:

1. Open a command prompt window by pressing Ctrl+Alt+T.
 2. In the command prompt window, then type “**cd [py folder location]**” and press Enter.
 - a. “cd” stands for change directory.
 - b. The “/opt/broadsim/SDX/API/Python” directory should be the folder location of all python script examples provided by Skydel as well as any python scripts that were created to automate testing.
- NOTE:** If new folders are created for the python script then the imports in the python script may need to be updated.
3. Type “**python [script name]**” and press Enter to execute the python script.
 - a. For an example, please see *Figure 4*.



```
Terminal
File Edit View Terminal Tabs Help
broadsim:~$ cd /opt/broadsim/SDX/API/Python
broadsim:/opt/broadsim/SDX/API/Python$ python advanced_gnss_spoofing_simulation_tutorial.py
```

Figure 4: Command prompt window executing the example script.

The script will then execute and print out like the first process. The only difference will be that the command prompt window will remain open until it is closed by the user. Keeping the command prompt window open will allow the user to more easily debug scripting issues that they may run into by providing script traceback information. See *Figure 5* for an example of how the script will look when it is executed using the command prompt window.

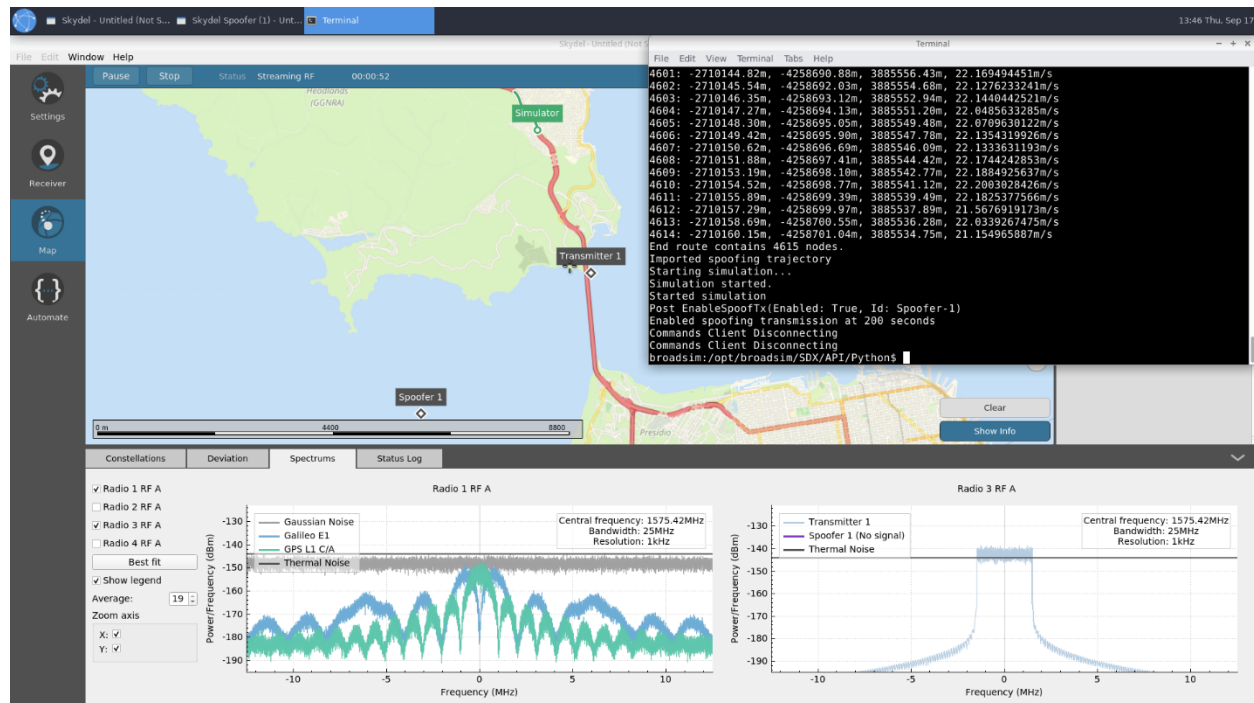


Figure 5: Example script executing and running on Skydel.